(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
2 May 2002 (02.05.2002)

PCT

(10) International Publication Number
WO 02/35755 A2

(51) International Patent Classification[7]: H04L

(21) International Application Number: PCT/US01/46080

(22) International Filing Date: 23 October 2001 (23.10.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/695,734 23 October 2000 (23.10.2000) US

(71) Applicant: **AKAMBA CORPORATION** [US/US]; 15055 Los Gatos Boulevard, Los Gatos, CA 95032 (US).

(72) Inventors: **WORLEY, W., Spencer, III**; 311 Correas Avenue, Half Moon Bay, CA 94019 (US). **VASTANO, John, A.**; 3497 Cabrillo Avenue, Santa Clara, CA 95051 (US). **MACDONELL, Eoin, B.**; 1057 Yorktown Drive, Sunnyvale, CA 94087 (US).

(74) Agent: **HENNEMAN, Larry, E. Jr.**; Henneman & Saunders, 121 E. 11th Street, Tracy, CA 95376 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 02/35755 A2

(54) Title: SYSTEM AND METHOD FOR HANDLING DENIAL OF SERVICE SERVER ATTACKS

(57) Abstract: A method for facilitating client (309) connections with a server (306) includes receiving a client connection request (502), determining whether a connection queue is full (504), clearing at least a portion of the connection queue if it is full (516), and placing the received client connection request (506). Optionally, a security routine is invoked (518) when it is determined that the connection queue (333) is full. In a particular embodiment, the methods of the present invention are performed by a queue-clearing communication module (422) for a proxy server (332) embedded in an adapter card (308) for an Internet web server.

SYSTEM AND METHOD FOR HANDLING DENIAL OF SERVICE SERVER ATTACKS

BACKGROUND

Field of the Invention

The present invention relates generally to network servers, and more particularly to a system and method for dealing with attacks on such servers. Even more particularly, the present invention relates to a system and method for dealing with denial of service attacks on network servers.

Description of the Background Art

Network servers (e.g., Internet web servers) provide information to clients (e.g., personal computers) over a network. A connection is established between the client computer and the server, and then the client and the server can exchange data (e.g., html web pages) via the connection. Typically, the client initiates the connection with the server.

Many servers provide information, entertainment, and services to members of the public via the Internet. However, opening a server to public access over the Internet makes the server vulnerable to malicious attacks. Such attacks have recently rendered servers popular Internet servers inoperable, costing the targeted companies thousands of dollars in lost sales, harming their corporate images, and requiring significant technical efforts to bring their servers back on line.

One type of server attack is a denial of service ("DOS") attack. This particular type of attack floods a server with more connection requests than the server can handle. Further, a DOS attack may take advantage of the three-way-hand-shake ("TWHS") process typically employed to establish a connection between a client and a server.

FIG. 1 is a transmission diagram 100 showing how the TWHS routine is used to establish a connection between a client and a server. The most common communication

protocol used on the Internet, TCP/IP, uses this type of TWHS routine to establish a connection between a client 102 and a server 104.

Typically, client 102 initiates the connection process by sending a connection request 106 to server 104. In response to the connection request 106, server 104 sends a connection request 108 including an acknowledgment of the connection request 106 sent by client 102. Finally, in response to the connection request and acknowledgment 108, client 102 sends an acknowledgment 110 of the connection request 108 sent by server 104. When server 104 receives acknowledgment 110, the connection is complete.

In the terminology of the TCP/IP protocol, a connection request is referred to as a "Syn" (synchronize sequence numbers flag). An acknowledgment is referred to as an "Ack." In one type of DOS attack (sometimes referred to as a Syn attack), one or more clients flood a server with Syns, but never respond with an ACK.

FIG. 2 is a block diagram of typical server communication software, and will help illustrate how DOS attacks (e.g., Syn attacks) prevent servers from accepting connections from authorized clients. Typical server software includes server applications 202, communications protocol stack 204, and connection queue 206. Server applications 202 communicates with clients (not shown in FIG. 2) via connections established by communications protocol stack 204. Connection queue 206 provides storage for pending connection requests ("PCRs"), and is conceptually divided into a completed connection queue of completed connection requests 208 and a an incomplete connection queue of incomplete connection requests 210. Those skilled in the art will understand that connection queue 206 need not be physically separated into a complete connection queue and an incomplete connection queue, but that the total number of PCRs, whether complete or incomplete) can not exceed some predetermined number (e.g., 1000).

Communications protocol stack 204 handles the TWHS routine of FIG. 1 as follows. When a client connection request is received, communications protocol stack 204 places the received connection request in connection queue 206 as an incomplete PCR 210, and sends a connection request and acknowledgment to the client. The incomplete PCR then remains in connection queue 206 until an acknowledgment is received from the client. When the acknowledgment is received from the client (i.e., TWHS complete), communication protocol stack 204 converts the incomplete PCR 210 into a complete PCR 208. Completed PCRs

typically remain in connection queue 206 only a very short time until they are accepted by server applications 202 and removed from the connection queue 206. If connection queue 206 is full, communication protocol stack 204 must ignore subsequent client connection requests.

In a Syn attack, the attacker sends multiple connection requests, but does not acknowledge the server's response (i.e., client sends Syns but does not Ack server Syns). Thus, connection queue 206 is filled with incomplete PCRs 210, and can no longer accept connection requests from authorized clients. Known communication protocol stacks have a time-out feature, whereby incomplete PCRs are automatically deleted if not acknowledged within a predetermined time interval (e.g., 75 seconds), but the time-out feature is inadequate to prevent the connection queue from being filled by a Syn attack. For example, a connection queue with a 1000 PCR capacity and a 75 second time-out can be filled by sending 13.33 syns per second. Further, the time-out period cannot be shortened too dramatically, because enough time must be allowed for authorized clients to complete the TWHS procedure. Additionally, dealing with the flood of connection requests is a burden on the server processing unit, and takes valuable processing cycles away from other server applications.

What is needed, therefore, is a communications module that facilitates authorized client connections in the face of a DOS attack. What is also needed is a communications module that can recognize a DOS attack and invoke security measures to deal with the attack. What is also needed is a system and method that takes the burden of dealing with a DOS attack off of the server's processing unit.

SUMMARY

The present invention overcomes the problems associated with the prior art by providing a communication module with connection queue clearing capabilities. The invention facilitates deleting pending connection requests from the connection queue during a denial-of-service attack, thereby providing a window of opportunity for authorized clients to establish a connection with the server.

One method of the present invention includes receiving a client connection request, determining whether a connection queue is full, clearing at least a portion of the connection queue if it is full, and placing the received client connection request in the connection queue.

In a particular embodiment, one or more incomplete connection request(s) are removed from the connection queue. In an alternate embodiment, one or more of the oldest connection request(s) are removed from the connection queue.

Optionally, a security routine is invoked when the it is determined that the connection queue is full. In one particular embodiment, the security routine is operative to clear redundant requests from the connection queue. In another particular embodiment, the security routine is operative to maintain security lists (e.g., authorized clients, known attackers, etc.), and to allow or remove redundant client connections based on those lists.

The present invention may be embodied in varied networking components, including but not limited to, primary servers, proxy servers, adapter cards, etc.. In a particular disclosed embodiment, a queue-clearing communication module and a proxy server are embedded in an adapter card for an Internet web server.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is described with reference to the following drawings, wherein like reference numbers denote substantially similar elements:

FIG. 1 is a transmission diagram of a TWHS routine for establishing a connection between a client and a server;

FIG. 2 is a block diagram showing typical server communication software;

FIG. 3 is a block diagram of a server including an adapter card with a queue-clearing communications protocol stack according to the present invention;

FIG. 4 is a block diagram showing one particular embodiment of the queue clearing protocol stack of FIG. 3 in greater detail; and

FIG. 5 is a flow chart summarizing one particular method of facilitating client connections with a server according to the present invention.

DETAILED DESCRIPTION

The present invention overcomes the problems associated with the prior art, by providing a communication module with connection queue clearing capabilities. In the following description, numerous specific details are set forth (e.g., connection queue capacity, particular protocols, etc.) in order to provide a thorough understanding of the invention.

Those skilled in the art will recognize, however, that the invention may be practiced apart from these specific details. In other instances, details of well known networking equipment and practices (e.g., physical media, data packet content, etc.) have been omitted, so as not to unnecessarily obscure the present invention.

FIG. 3 is a block diagram showing a system 300 coupled to an internetwork 302 via a physical network media 304. In a particular implementation, system 300 is an Internet web server, and internetwork 302 is the Internet, but those skilled in the art will recognize that the present invention may be implemented in any type of network server.

System 300 includes a file server (e.g., an HTTP web server) 306 and an adapter card 308. File server 306 provides data to and receives data from clients 309 on internetwork 302, via adapter card 308. Adapter card 308 establishes and maintains network connections between clients 309 and adapter card 308, and establishes bus connections between server 306 and adapter card 308. Thus connected, adapter card 308 receives communications from clients 309 on behalf of server 306, forwards the communications to server 306, receives responses from server 306 on behalf of clients 309, and forwards the responses to clients 309.

Server 306 includes non-volatile memory 310, working memory 312, server mass data storage 314, a processing unit 316, and one or more user input/output (I/O) devices 318, all intercommunicating via a server bus 320 (e.g., PCI bus). Non-volatile memory 310 (e.g., read-only memory and/or one or more hard-disk drives) provides storage for data and code which is retained even when server 306 is powered down. Working memory 312 (e.g., random access memory) provides operational memory for server 306, and includes executable code (e.g., an operating system) which is loaded into working memory 312 during start-up. Among other programs, working memory 312 includes server applications 321 and a communication protocol stack 322. Server applications 321 include network software applications (e.g., FTP, HTTP, etc.) which allow server 306 to function as a network server. Communications protocol stack 322 is a standard protocol stack (e.g., TCP/IP) which facilitates communication with other machines over an internetwork. Standard protocol stacks are well known in the art. See, for example, W. Richard Stevens, *TCP/IP Illustrated, Vol. 1* (Addison-Wesley, 1994), which is incorporated herein by reference. Server mass data storage 314 provides data storage (e.g., one or more hard disk drives) for data (e.g., HTML pages, graphics files, etc.), which the server provides to clients 309 attached to internetwork

302. Processing unit 316 executes the instructions in working memory 312 to cause server 306 to carry out its primary function (e.g., providing data to and receiving data from clients). I/O devices 318 typically include a keyboard, a monitor, and/or such other devices which facilitate user/administrator interaction with server 306. Each of the above described components is typically found in a network server such as an Internet web server.

Adapter card 308 includes non-volatile memory 323, working memory 324, a processing unit 326, a bus protocol bridge 328, and a network controller 329, all intercommunicating via an adapter bus 330. Non-volatile memory 323 provides storage for data and code (e.g., boot code) which is retained even when adapter 308 is powered down. Processing unit 326 imparts functionality to adapter card 308 by executing the code present in working memory 324. Bus protocol bridge 328 provides an interface between adapter bus 330 and server bus 320, and network controller 329 provides an interface between adapter bus 330 and network media 304.

Working memory 324 provides operational memory for adapter 308, and includes a proxy application 332, a connection queue 333, and a queue-clearing communication module 334. Proxy 332 and communication module 334 are loaded from non-volatile memory 323 into working memory 324 at start-up, and connection queue is 333 is generated by communication module 334 during operation. Optionally, proxy 332 and protocol stack 334 can be loaded from one or more alternative sources, including but not limited to non-volatile memory 310 or server mass data storage 314 of server 306. Proxy 332, when executed by processing unit 326, accepts and manages the above described connections between adapter 308 and server 306 and between adapter 308 and clients 309, via queue-clearing communication module 334.

In this particular embodiment of the invention, protocol stack 322 is a standard (e.g., TCP/IP) protocol stack, and communication module 334 is a standard protocol stack that has been modified according to the present invention to have queue clearing capabilities. Using a modified version of a standard communication protocol stack in adapter 308 facilitates the use of the standard communication software (e.g., protocol stack 322) already present in the vast majority of network servers. Those skilled in the art will recognize, however, that this particular element (as well as other described elements, even if not explicitly stated) is not an essential element of the present invention. For example, the present invention may be

practiced with custom communication software (e.g., direct communication between server applications 321 and either communications module 334 or proxy 332) in both server 306 and adapter 308. Further, in particular embodiments of the invention, this element may be omitted by providing proxy 332 with direct access to the resources (e.g., server mass data storage 314) of server 306. As yet another example, queue-clearing communication module 334 and connection queue 333 can be employed directly in working memory 312 of server 306, without a proxy application or protocol stack in adapter card 308. As even yet another example, proxy 332, connection queue 333 and queue clearing communication module 334 can be embodied in a proxy server separate from server 306 that communicates with server 306 via a network connection. This list of examples is provided to illustrate the great flexibility of the present invention, and is no way intended to be or considered to be a complete list of possible embodiments of the present invention.

Adapter card 308 is coupled to server 306 via a bus connection 336 between bus protocol bridge 326 and server bus 320. In this particular embodiment, bus connection 336 is a typical bus expansion slot, for example a PCI slot. Those skilled in the art will recognize, however, that the present invention may be implemented with other types of bus connections, including but not limited to an ISA slot, a USB port, a serial port, or a parallel port. Bus connection 336 facilitates high speed, large packet size, relatively error free (as compared to network connections) communication between proxy 332 and server applications 321, greatly reducing the connection management burden on processing unit 316 of server 306. In addition, in this particular embodiment processing unit 326 of adapter card 308 handles the burden of dealing with DOS attacks, freeing processing unit 316 to carry out the primary functions of server 306. In summary, proxy 332 (running on processing unit 326) communicates with clients 309 over slow, error prone network connections, and then communicates with server applications 321 on behalf of clients 309 over high speed bus connection 336. Further, queue-clearing communication module 334 runs on processing unit 326 and establishes connections with clients 309 even when under a DOS attack, as will be described below with reference to FIG. 4.

FIG. 4 is a block diagram of working memory 324 showing connection queue 333 and queue-clearing communication module 334 in greater detail. Those skilled in the art will recognize that while the various software modules of communication module 334 are shown

as interconnected functional blocks, the software modules are actually blocks of executable code stored in working memory 324 that can communicate with one another when executed by processing unit 326 (FIG. 3).

Further, connection queue 333 includes both incomplete connection requests 426 and complete connection requests 428, but need not necessarily be a particular physical portion of working memory 324. More likely, connection queue 333 is a list generated by communication module 334 with a predetermined maximum number of pending connection requests (PCRs). The list includes both complete PCRs and incomplete PCRs, and can be conceptually divided into a completed connection queue and an incomplete connection queue, but the total number of PCRs cannot exceed the predetermined capacity of connection queue 333.

In this particular embodiment of the present invention, queue-clearing communication module 334 is a modified TCP/IP stack including a sockets layer 410, a modified TCP layer 412, an IP layer 414, and a device layer including a network driver 416 and a server bus driver 418. The functionality of each of the individual layers of protocol stack 334, except for modified TCP layer 412, is well known in the art, and will not, therefore, be discussed in detail herein.

Modified TCP layer 412 includes a transmission control (TC) module 420, a clear queue module 422, and a security module 424. TC module 420 is functionally similar to a conventional TCP layer except that TC module 420 has the ability to call one or both of clear queue module 422 and security module 424 when a client connection request is received and connection queue 333 is full.

Clear queue module 422, when called by TC module 420, clears at least a portion of connection queue 333 by removing at least one PCR from connection queue 333, and sending a reset signal to the client 309 (FIG. 3) associated with the deleted PCR. In this particular embodiment of the invention, clear queue module 422 deletes the oldest, incomplete PCR from connection queue 333. Those skilled in the art will recognize, however, that alternate queue-clearing schemes may be employed, including but not limited to deleting multiple PCRs from connection queue 333 or deleting PCRs based on some criteria (e.g., source IP address, etc.) other than time in connection queue 333.

Security module 424 is a general purpose security module that is called by TC module 420 when TC module 420 determines that connection queue 333 is full. Alternatively, security module 424 periodically scans connection queue 333 and is self-executing responsive to connection queue 333 being full. In either case, once activated security module 424 filters incoming client connection requests based on some criteria (e.g., source IP). For example, all incoming connection requests from a particular source (i.e., an attacker) are ignored.

Queue-clearing communications module 412 establishes connections between proxy 332 and clients 309 as follows. When TC module 420 receives a connection request from a client 309, TC module 420 places the connection request in connection queue 333 as an incomplete PCR 426 and transmits a Syn/Ack to the associated client 309. Then, when TC module 420 receives the ACK (completes TWHS) from the client, the incomplete PCR 426 is converted to a complete PCR 428, which is then accepted by proxy 332 and removed from the connection queue 333. If connection queue 333 is full (already has maximum number of entries) when a client connection request is received, then TC module 420 calls clear queue module 422 to clear at least a portion of connection queue 333 to make room for the incoming client connection request. Then, TC module 420 places the incoming client connection request in connection queue 333.

Clearing a portion (e.g., the oldest incomplete PCR) of connection queue 333 to make room for an incoming connection requests gives authorized clients a window of opportunity to establish a connection with proxy 332, and requires that an attacker generate a higher frequency of connection request to block connections with proxy 332. For example, in a conventional protocol stack, a denial of service occurs when the time-out period multiplied by the connection request arrival rate is greater than the capacity of the connection queue. For a connection queue capacity of 1,000 PCRs and a time-out period of 75 seconds, an attacker would only need to generate 13.33 connection requests per second to cause a denial of service.

According to the present invention, however, queue clearing communication module 334 continues to accept incoming connection requests, clears at least one PCR from the connection queue, and then places the incoming connection request in the connection queue. The result is that all incoming connection requests are cycled through the connection queue, giving authorized clients a window of opportunity (i.e., the time their PCR is in connection

queue 422) to complete the TWHS and completed the PCR. As long as clear module 422 only clears incomplete PCRs, the completed PCR would not necessarily have to be accepted by proxy 332 within the window of opportunity. The round-trip-time (RTT) for a connection is the sum of the time required for the server's connection request to reach the client and the client's acknowledgment to reach the server. The RTT for most internetwork connections today is less than 250 milliseconds. Therefore, as long as the authorized client's PCR is in the connection queue for at least 250 msec., it is likely that the connection will be completed.

In order for an attacker cause a denial of service on a server implementing the present invention, the connection request arrival rate would need to be sufficiently high to reduce the time an incomplete PCR 426 spends in connection queue 333 below the RTT of a connection. For example, the time that an incomplete PCR remains in the connection queue is equal to the capacity of connection queue 333 divided by the connection request arrival rate. For the example connection queue with a 1,000 entry capacity and a 250 msec RTT, an attacker would need to generate 4,000 connection requests per second to cause a denial of service. This rate is 300 times greater that the rate (13.33 connection requests per second) required for a denial of service on the conventional protocol stack of the previous example. In other words, this example embodiment of the present invention provides 300 times the protection afforded by the prior art systems.

FIG. 5 is a flow chart summarizing one particular method 500 of facilitating connections between a client and a server according to the present invention, and will be described with reference to the example embodiment shown in FIGs. 3-4. The method of the present invention is not, however, intended to be in any way limited to the particular embodiment shown in FIGs. 3-4. In fact, it is anticipated that the method of the present invention will be useful in a wide variety of systems, including but not limited to stand-alone servers, proxy servers, adapter cards, etc..

In a first step 502, TC module 420 receives a connection request (Syn) from a client 309, and in a second step 504 determines whether connection queue 333 is full (e.g., maximum number of PCRs). If connection queue 333 is not full, then in a third step 506, TC module 420 places the connection request in connection queue 333 as an incomplete PCR 426 and sends a request/acknowledgment (Syn/Ack) to client 309. Next, in a fourth step 508, TC module 420 determines whether client 309 has acknowledged the incomplete PCR 426. If

client 309 has not acknowledged incomplete PCR 426, then in a fifth step 510, TC module 420 determines whether the incomplete PCR 426 has timed out (i.e., been in connection queue 333 longer than a predetermined time limit). If the incomplete PCR has timed out, then in a sixth step 512 TC module 420 sends a reset signal to client 309, and in a seventh step 514 deletes incomplete PCR 426 from connection queue 333.

If, in second step 504, TC module 420 determines that connection queue 333 is full, then method 500 proceeds to a eighth step 516 wherein TC module 420 deletes the oldest, incomplete PCR from connection queue 333. Next, in an ninth step 518, TC module 420 calls security module 424 to implement a security routine (e.g., source IP address filtering of subsequent incoming connection requests). Then method 500 returns to third step 506.

If, in fourth step 508, TC module 420 determines that acknowledgment of the incomplete PCR has been received from client 309, then method 500 proceeds to a tenth step 520, wherein TC module 420 converts incomplete PCR 426 into a complete PCR 428. Next, in an eleventh step 522, TC module 420 determines whether completed connection PCR has been accepted by proxy 332. If not, eleventh step 522 is periodically repeated, until complete PCR 428 is accepted by proxy 332. Then method 500 proceeds to seventh step 514.

The description of particular embodiments of the present invention is now complete. Many of the described features may be substituted, altered or omitted without departing from the scope of the invention. For example, the queue-clearing communication module described herein need not be implemented in a TCP/IP protocol stack, but may be implemented in any communication software that establishes and or manages connections between clients and a server. Additionally, the present invention may be embodied in different networking components, including but not limited to a primary server, a proxy server, an adapter card, etc.. These and other deviations from the particular embodiments shown will be apparent to those skilled in the art, particularly in view of the foregoing disclosure.

We claim:

1. A method for facilitating client connections with a server comprising:
receiving a client connection request;
determining whether a connection queue is full;
clearing at least a portion of said connection queue if said connection queue is full; and
placing said client connection request in said connection queue.

2. A method according to Claim 1, wherein said step of clearing at least a portion of said connection queue comprises deleting one connection request from said connection queue.

3. A method according to Claim 2, wherein said step of deleting one connection request from said connection queue comprises deleting only an incomplete connection request.

4. A method according to Claim 3, wherein said step of deleting only an incomplete connection request from said connection queue comprises deleting the oldest incomplete connection request from said connection queue.

5. A method according to Claim 2, wherein said step of deleting one connection request from said connection queue comprises deleting the oldest connection request from said connection queue.

6. A method according to Claim 2, wherein said step of deleting one connection request from said connection queue comprises sending a reset signal to a client associated with said deleted connection request.

7. A method according to Claim 1, further comprising invoking a security routine if it is determined that said connection queue is full.

8. A method according to Claim 7, wherein said security routine comprises filtering subsequent incoming client connection requests based on their source IP addresses.

9. A method according to Claim 1, wherein said step of receiving said client connection request comprises receiving said client connection request as a proxy on behalf of said server.

10. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 1.

11. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 2.

12. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 3.

13. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 4.

14. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 5.

15. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 6.

16. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 7.

17. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 8.

18. A computer-readable medium having code embodied therein for causing a computer to perform the method of Claim 9.

19. A server for establishing connections with clients on a network, said server comprising:

a network controller for receiving client connection requests via said network;

a memory device for storing data and code, said code including a connection queue and a communications module, said communications module being responsive to receipt of a client connection request and operative to clear at least a portion of said connection queue if said connection queue is full and to place said client connection request in said connection queue; and

a processing unit coupled to said memory device and said network controller for executing said code.

20. A server according to Claim 19, wherein said communications module responsive to receipt of said client connection request is operative to delete one connection request from said connection queue if said connection queue is full.

21. A server according to Claim 20, wherein said communications module responsive to receipt of said client connection request is operative to delete an incomplete connection request from said connection queue if said connection queue is full.

22. A server according to Claim 21, wherein said communications module responsive to receipt of said client connection request is operative to delete the oldest incomplete connection request from said connection queue if said connection queue is full.

23. A server according to Claim 20, wherein said communications module responsive to receipt of said client connection request is operative to delete the oldest connection request from said connection queue if said connection queue is full.

24. A server according to Claim 20, wherein said communications module is further operative to transmit a reset signal to a client associated with said deleted connection request.

25. A server according to Claim 19, wherein:

said memory device further comprises a security module; and

said security module responsive to said connection queue being full is operative to invoke a security routine.

26. A server according to Claim 25, wherein said security routine includes clearing redundant connection requests from said connection queue.

27. A server according to Claim 26, wherein said security routine includes:

maintaining a list of authorized clients; and

allowing redundant connection requests from said authorized clients to remain in said connection queue while clearing other redundant connection requests from said connection queue.

28. A server according to Claim 25, wherein said security routine comprises filtering incoming connection requests based on their source IP addresses.

29. A server according to Claim 19, wherein:

said server is a proxy server for a second server; and

said server further includes an adapter to facilitate communication with said second server.

30. A server according to Claim 29, wherein:

said server is embodied in an adapter card; and

said adapter facilitates a bus connection with said second server.

31. A server according to Claim 19, wherein:

said communications module is embodied in an adapter card; and

said adapter card includes an adapter to facilitate a bus connection with said server.

32. A server according to Claim 19, wherein said communications module is operative to clear redundant connection requests from said connection queue.

33. A server according to Claim 32, wherein said communications module is operative to:

maintain a list of authorized clients; and

allow redundant connection requests from said authorized clients to remain in said connection queue while clearing other redundant connection requests from said connection queue.

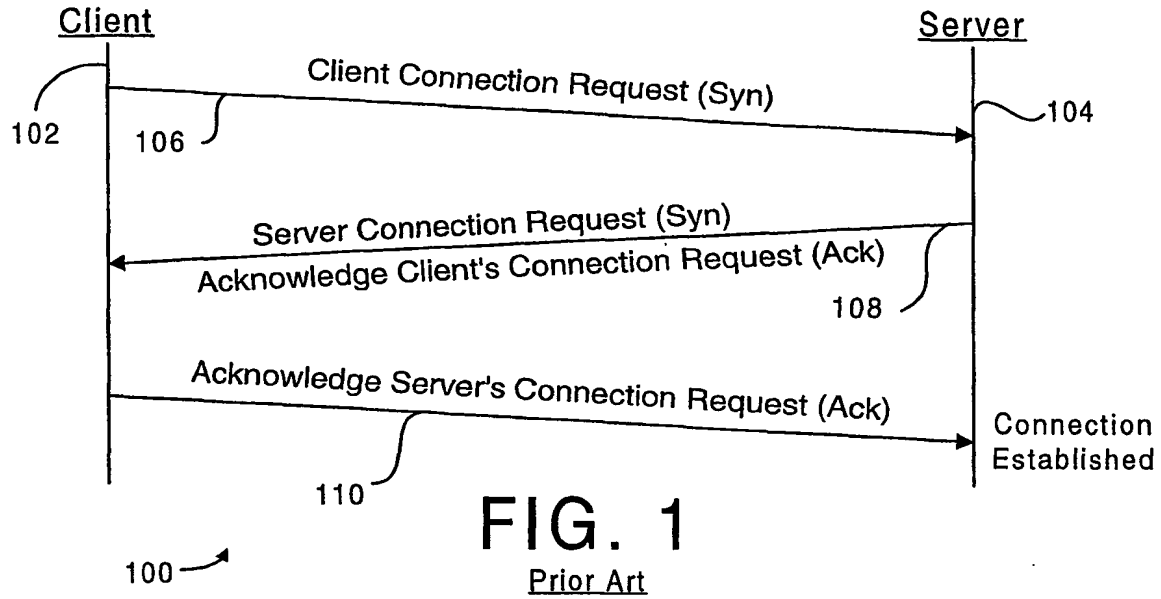

FIG. 1
Prior Art

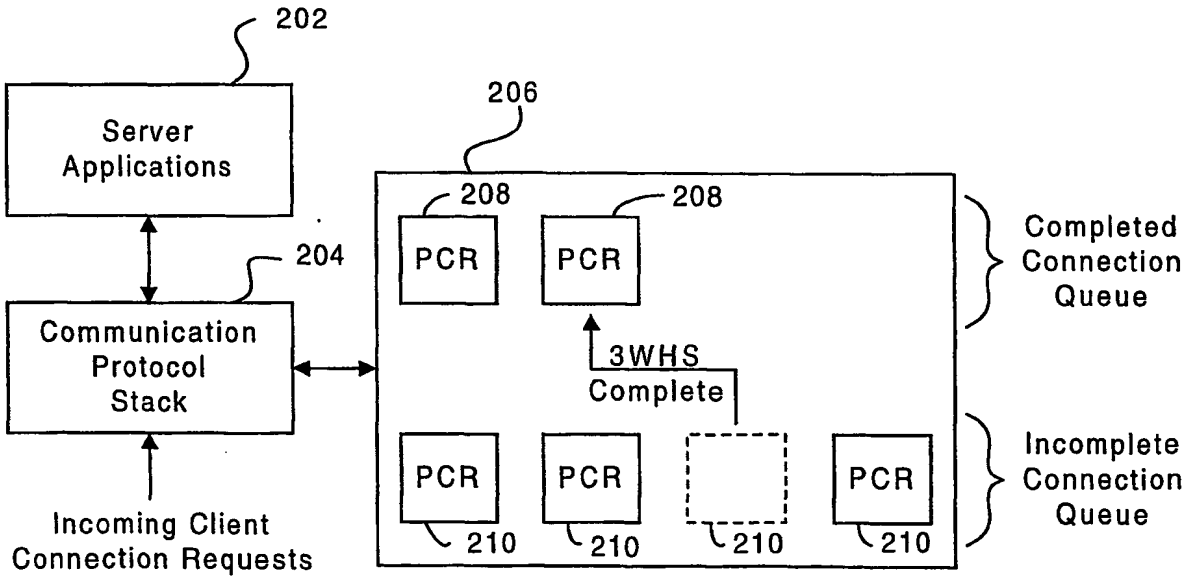

FIG. 2
Prior Art

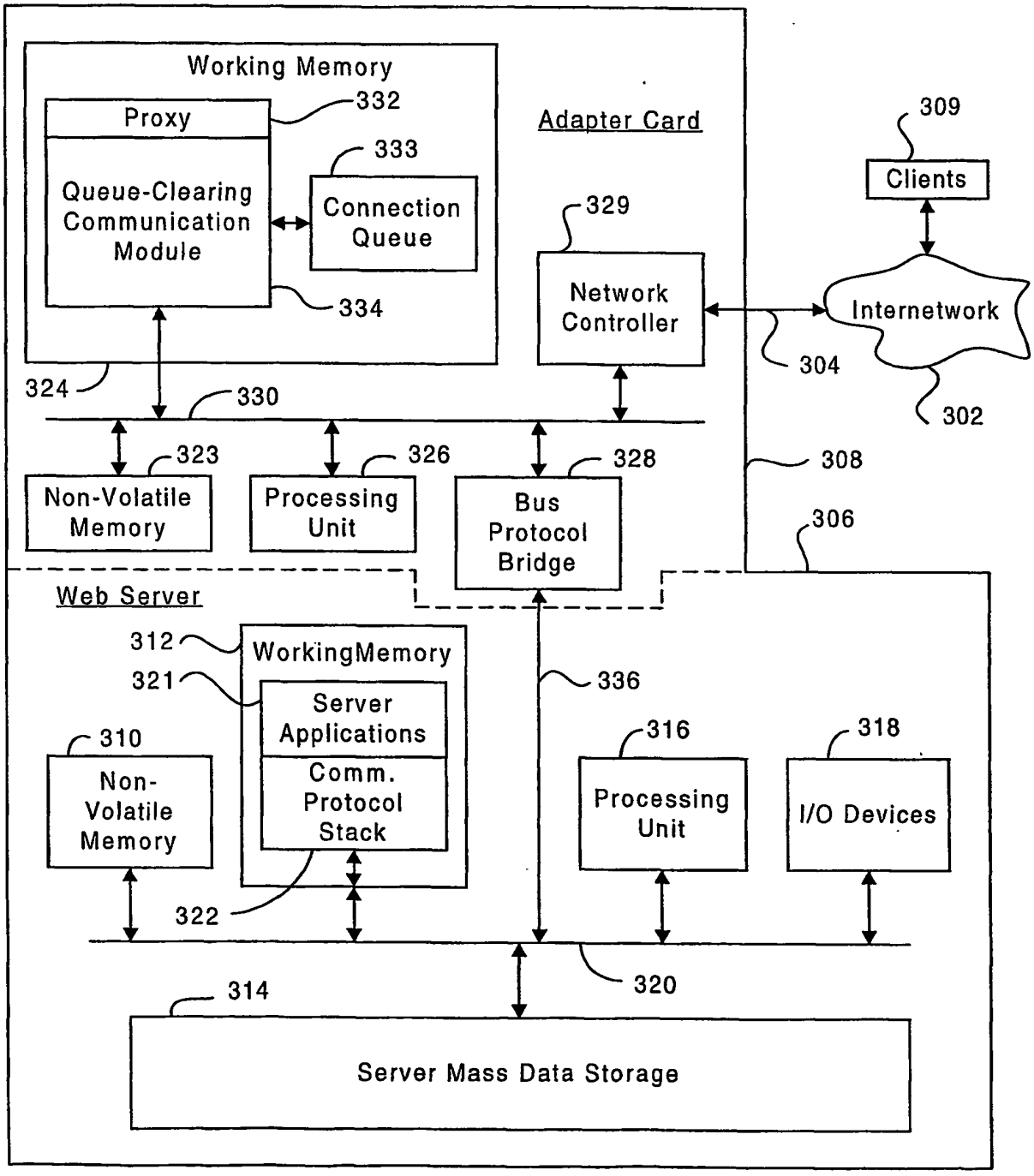
Working Memory
Proxy
332
Queue-Clearing Communication Module
333
Connection Queue
334
Adapter Card
309
Clients
329
Network Controller
Internetwork
304
302
324
330
323
326
328
308
Non-Volatile Memory
Processing Unit
Bus Protocol Bridge
306
Web Server
312
WorkingMemory
321
Server Applications
Comm. Protocol Stack
336
310
Non-Volatile Memory
316
Processing Unit
318
I/O Devices
322
320
314
Server Mass Data Storage
300

FIG. 3

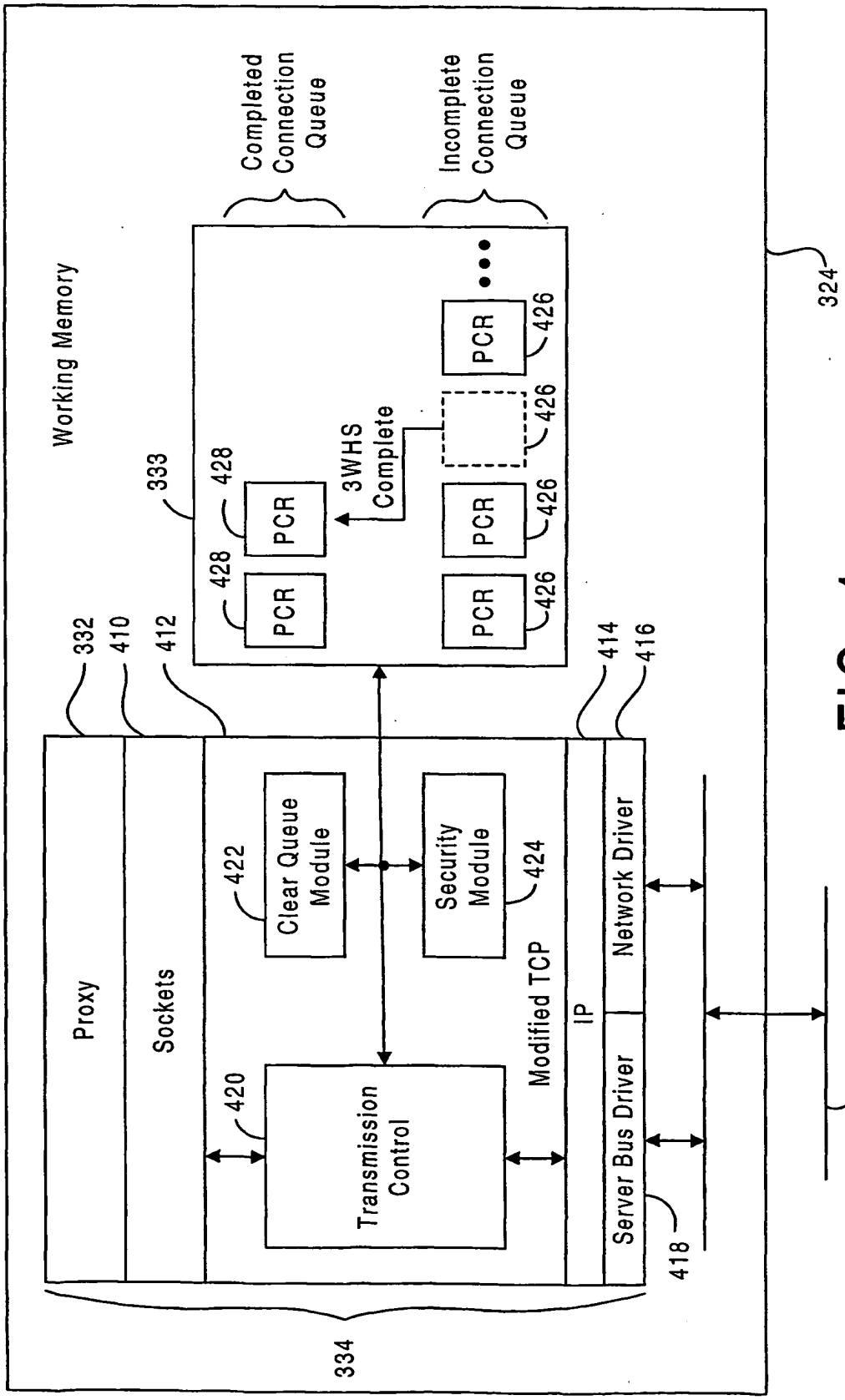
Working Memory
Completed Connection Queue
Incomplete Connection Queue
333
428
PCR
3WHS Complete
426
332
410
412
414
416
Proxy
Sockets
Clear Queue Module
422
Security Module
424
Modified TCP
IP
Network Driver
Server Bus Driver
Transmission Control
420
418
334
330
324

FIG. 4

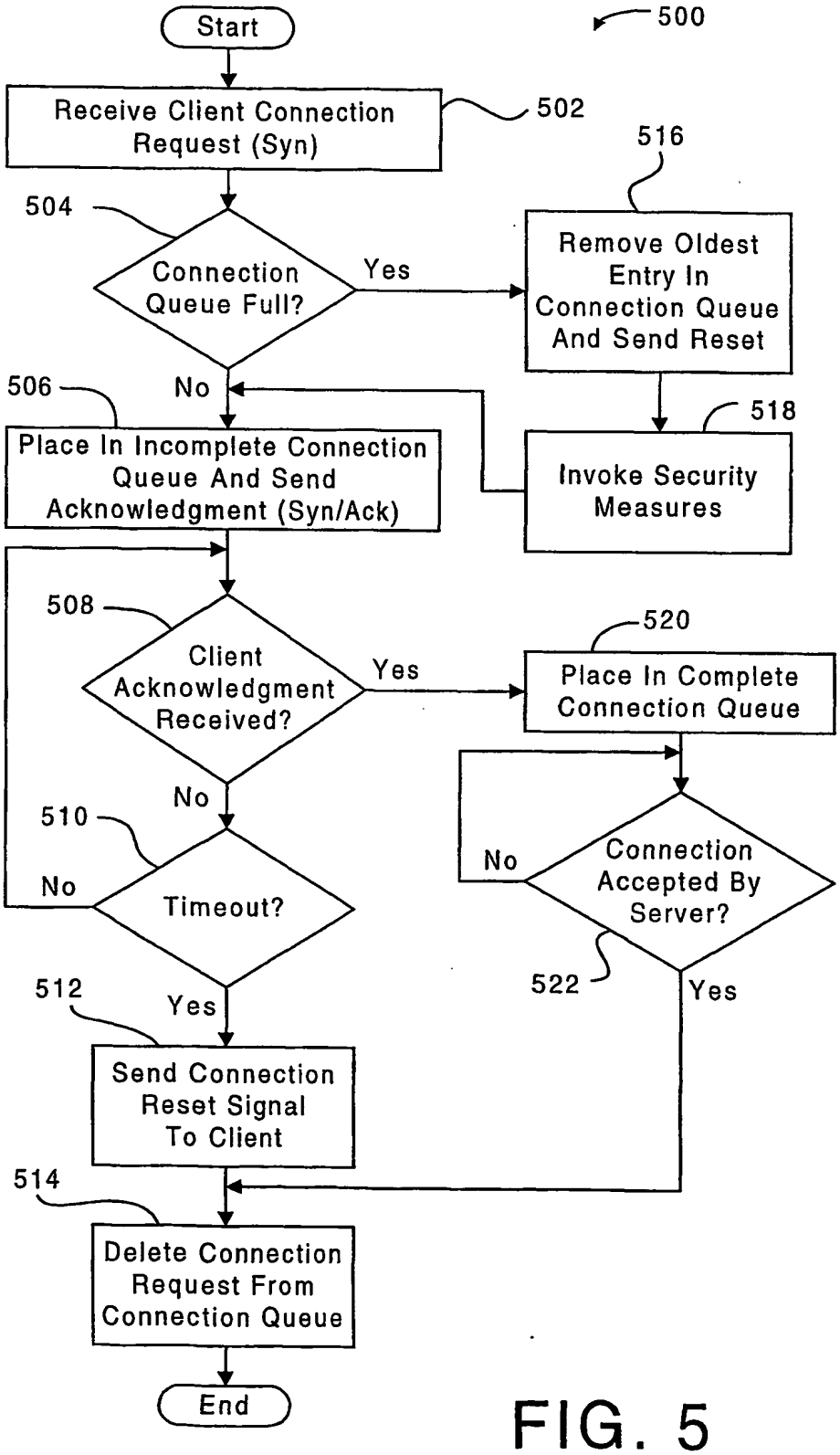
Start
500
Receive Client Connection Request (Syn)
502
504
Connection Queue Full?
Yes
516
Remove Oldest Entry In Connection Queue And Send Reset
506
No
Place In Incomplete Connection Queue And Send Acknowledgment (Syn/Ack)
518
Invoke Security Measures
508
Client Acknowledgment Received?
Yes
520
Place In Complete Connection Queue
No
510
No
Timeout?
No
Connection Accepted By Server?
522
Yes
512
Yes
Send Connection Reset Signal To Client
514
Delete Connection Request From Connection Queue
End

FIG. 5